

JTRS Have Quick MAC API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release
-----	-----------------

Table of Contents

1	INTRODUCTION.....	1
1.1	OVERVIEW	1
1.2	SERVICE LAYER DESCRIPTION.	1
1.3	MODES OF SERVICE.....	2
1.4	SERVICE STATES.	2
1.5	REFERENCED DOCUMENTS.....	2
2	UUID.....	2
3	SERVICES.....	3
3.1	MAC COMMON UTILITY.....	4
3.1.1	activateChannel Service.	4
3.1.2	setRFPowerControl.	4
3.1.3	setMode Service.	4
3.1.4	getMinTU Service.....	4
3.1.5	getMaxTU Service.	4
3.1.6	transmitTOD Service.....	5
3.1.7	receiveTOD Service.....	5
3.1.8	stopReceiveTOD Service.	6
3.1.9	setXmitMode Service.....	6
3.2	TRANSEC.....	7
3.2.1	loadFill Service.	7
3.2.2	loadSeed Service.	7
3.2.3	readSeed Service.	8
3.2.4	zeroize Service.	8
3.2.5	loadNetNumber Service.	9
3.2.6	readNetNumber Service.	9
3.2.7	loadFMTFrequencies Service.	10
3.2.8	readFMTFrequencies Service.	10
3.3	CHANNELERRORCONTROL.....	11
3.3.1	setChannelErrorControl Service.	11
3.4	ULONGPACKET.	12
3.4.1	pushPacket Control Service.	12
4	SERVICE PRIMITIVES.....	13
4.1	MAC COMMON UTILITY SERVICE PRIMITIVES.	13
4.1.1	activateChannel Service.	13
4.1.2	setRFPowerControl Service.	13
4.1.3	setMode Service.	14
4.1.4	getMinTU Service.....	15
4.1.5	getMaxTU Service.	15
4.1.6	transmitTOD Service.....	16
4.1.7	receiveTOD Service.....	16
4.1.8	stopRecTOD Service.....	17
4.1.9	setXmitMode Service.....	17

4.2	TRANSEC SERVICE PRIMITIVES.....	18
4.2.1	loadFill Service.....	18
4.2.2	loadSeed Service.....	19
4.2.3	readSeed Service.....	19
4.2.4	zeroize Service.....	20
4.2.5	loadNetNumber Service.....	21
4.2.6	readNetNumber Service.....	21
4.2.7	loadFMTFrequencies Service.....	22
4.2.8	readFMTFrequencies Service.....	23
4.3	CHANNELERRORCONTROL SERVICE PRIMITIVES.....	23
4.3.1	setChannelErrorControl Service.....	23
4.4	ULONGPACKET SERVICE PRIMITIVES.....	24
4.4.1	pushPacket Service.....	24
5	ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.....	25
6	PRECEDENCE OF SERVICE PRIMITIVES.....	25
7	SERVICE USER GUIDELINES.....	25
8	SERVICE PROVIDER-SPECIFIC INFORMATION.....	25
9	IDL.....	25
10	UML.....	38
10.1	CONTROLLER DIAGRAM.....	38
10.2	MAC COMMON UTILITY DIAGRAM.....	39
10.3	TRANSEC DIAGRAM.....	40
10.4	CHANNELERRORCONTROL DIAGRAM.....	41
10.5	USER DIAGRAM.....	42
10.6	PROVIDER DIAGRAM.....	42
10.7	ULONGPACKET DIAGRAM.....	43
10.8	COMPONENT DIAGRAM.....	44

List of Figures

Figure 1. MAC Common Utility.....	4
Figure 2. Sequence Diagram, MAC Common Utility.....	5
Figure 3. TRANSEC	7
Figure 4. Sequence Diagram, loadFill.....	7
Figure 5. Sequence Diagram, loadSeed and readSeed.....	8
Figure 6. Sequence Diagram, zeroize.....	8
Figure 7. Sequence Diagram, loadNetNumber and readNetNumber.....	9
Figure 8. Sequence Diagram, loadFMTFrequencies and readFMTFrequencies	10
Figure 9. Channel Error Control.....	11
Figure 10. Sequence Diagram, setChannelErrorControl.....	11
Figure 11. Data Packet.....	12
Figure 12. Sequence Diagram, Real-Time pushPacket.....	12
Figure 13. UML Controller Relationships	38
Figure 14. UML MAC Common Utility Relationships	39
Figure 15. UML TRANSEC Relationships.....	40
Figure 16. UML Channel Error Control Relationships.....	41
Figure 17. UML User, Data Packet Relationships.....	42
Figure 18. UML Provider, Data Packet Relationships.....	42
Figure 19. UML UlongPacket Relationships	43
Figure 20. HQ Component Diagram.....	44

List of Figures

Table 1. Cross-Reference of Services and Primitives.....	3
--	---

1 INTRODUCTION.

1.1 OVERVIEW.

Media Access Control (MAC) application-program interfaces (APIs) provide standardized interfaces to the MAC Layer. This MAC API is intended to support the Have Quick API.

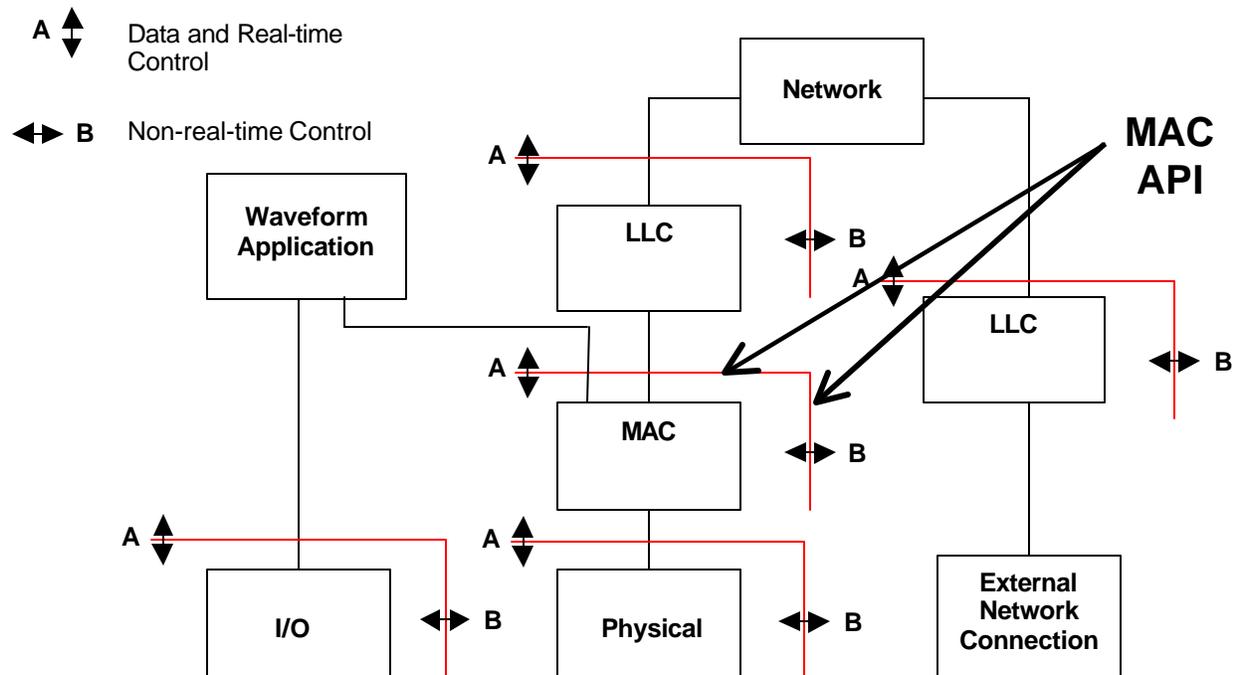
1.2 SERVICE LAYER DESCRIPTION.

The primary Service used by the MAC is the Physical Layer. The MAC Service Layer is subdivided into the following Service Groups/ BBs:

- MAC Common Utility
- TRANSEC
- Channel Error Control
- Channel Access
- MAC Address
- Drop Capture
- Quality of Service (QOS)
- Packet (Provided by the Packet BB API).

The HQ API utilizes the MAC Common Utility, the TRANSEC, the Channel Error Control, and the Packet (Provided by the Packet BB API) service groups.

The Location of the MAC Layer with respect to the other layers is shown in the following diagram.



1.3 MODES OF SERVICE.

There are no specific Modes of Service. MAC mode services are operation-oriented and support control-data transfer in self-contained units with no logical relationships required between BBs.

1.4 SERVICE STATES.

Each building block realization is an instantiation of a parameterized building block, which subsumes the states enumerated in a waveform-specific concrete building block.

1.5 REFERENCED DOCUMENTS.

<u>Document No.</u>	<u>Document Title</u>
MSRC-5000SCA	Software Communications Architecture Specification
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix F Media Access control Building Block Service Definition
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix D Physical Real-Time Building Block Service Definition

2 UUID.

The UUID for this API is 55b098e0-d1d3-11d4-8cc8-00104b23b8a2.

3 SERVICES.

A cross-reference of Services and Primitives provided by the MAC Layer is provided in Table 1.

Table 1. Cross-Reference of Services and Primitives

Service Group	Service	Primitives
MACCommon Utility	Activate Channel	activateChannel (PresetNum : in short) : boolean
	Set RF Power Control	setRFPowerControl (PoweLevel : in PowerModeType) : boolean
	Set Mode	setMode (ChannelMode : in ChannelModeType) : boolean
	Get Minimum Tx Unit Length	getMinTU (MinTU : out unsigned long) ; void
	Get Maximum Tx Unit Length	getMaxTU (MaxTU : out unsigned long) ; void
	Transmit TOD	transmitTOD(): boolean
	Receive TOD	receiveTOD(): boolean
	Stop Receive TOD	stopRecTOD(): boolean
	Set Transmit Mode	setXmitMode (PTTIndicator : in boolean) : boolean
TRANSEC	Load Fill	loadFill (presetNum : in short, MWODFill : in MWODFillType) : boolean
	Load Seed	loadSeed (SeedNum : in short = NULL, TODSeed : in TODSeedType) : boolean
	Read Seed	readSeed (SeedNum : in short = NULL, TODSeed : out TODSeedType) : boolean
	Zeroize Fill/Seed	zeroize (): boolean
	Load Net Number	loadNetNumber (PresetNum: in short, netData: in NetNumberType) : boolean
	Read Net Number	readNetNumber (PresetNum: in short, netData: out NetNumberType) : boolean
	Load FMT Frequencies	loadFMTFrequencies (FMTFill : in FMTFillType) : boolean
	Read FMT Frequencies	readFMTFrequencies (FMTFill : out FMTFillType) : boolean
Channel Error Control	Enable/Disable Error Control	setChannelErrorControl (ErrorControl : in ErrorControlType) : void
Transmit / Receive Packets	Packet BB Instantiated only. Not defined in this API. See Service Definition Description Generic Packet Service Building Block	pushPacket(priority : in octet, control : in HQ_API::HQ::NULLControl, payload : in HQ_API::PortTypes::UlongSequence)
	NullControl	Null

3.1 MAC COMMON UTILITY.

The MAC Common Utility Service Group (Figure 1) provides generalized interfaces to the MAC layer for methods required by more than one MAC Building Block (BB). Refer to Figure 2 for the following discussion of sequences.

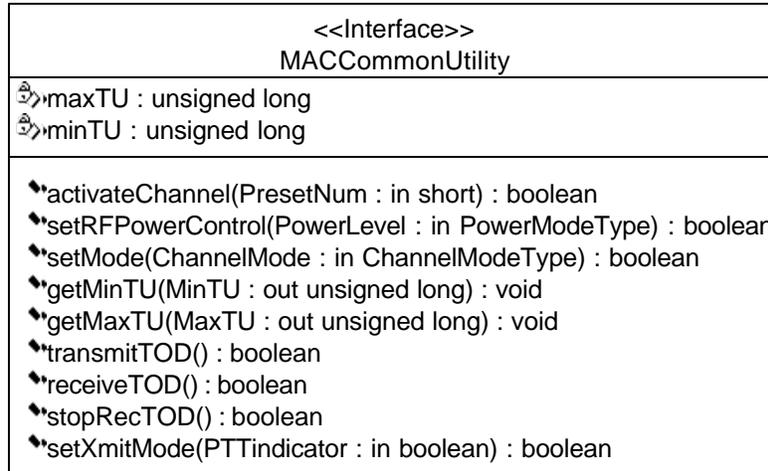


Figure 1. MAC Common Utility

3.1.1 activateChannel Service.

The activateChannel Service provides Service Users with a method to communicate to the MAC Layer the active radio channel. The specified preset parameters are activated.

3.1.2 setRFPowerControl.

The setRFPowerControl Service provides Service Users with a method to communicate to the MAC Layer the selected RF Power Control mode. Power mode can be used to designate a specific level of RF power output or to designate the RF power level will be controlled automatically by algorithms contained within the MAC Layer or within data packets.

3.1.3 setMode Service.

The setMode Service provides Service Users with a method to communicate to the MAC Layer the selected waveform.

3.1.4 getMinTU Service.

Provides Service Users with the minimum Transmission Unit (TU) length the Service Provider will accept as being a valid packet. This is a read-only method.

3.1.5 getMaxTU Service.

Provides Service Users with the maximum Transmission Unit (TU) length the Service Provider will accept as being a valid packet. This is a read-only method.

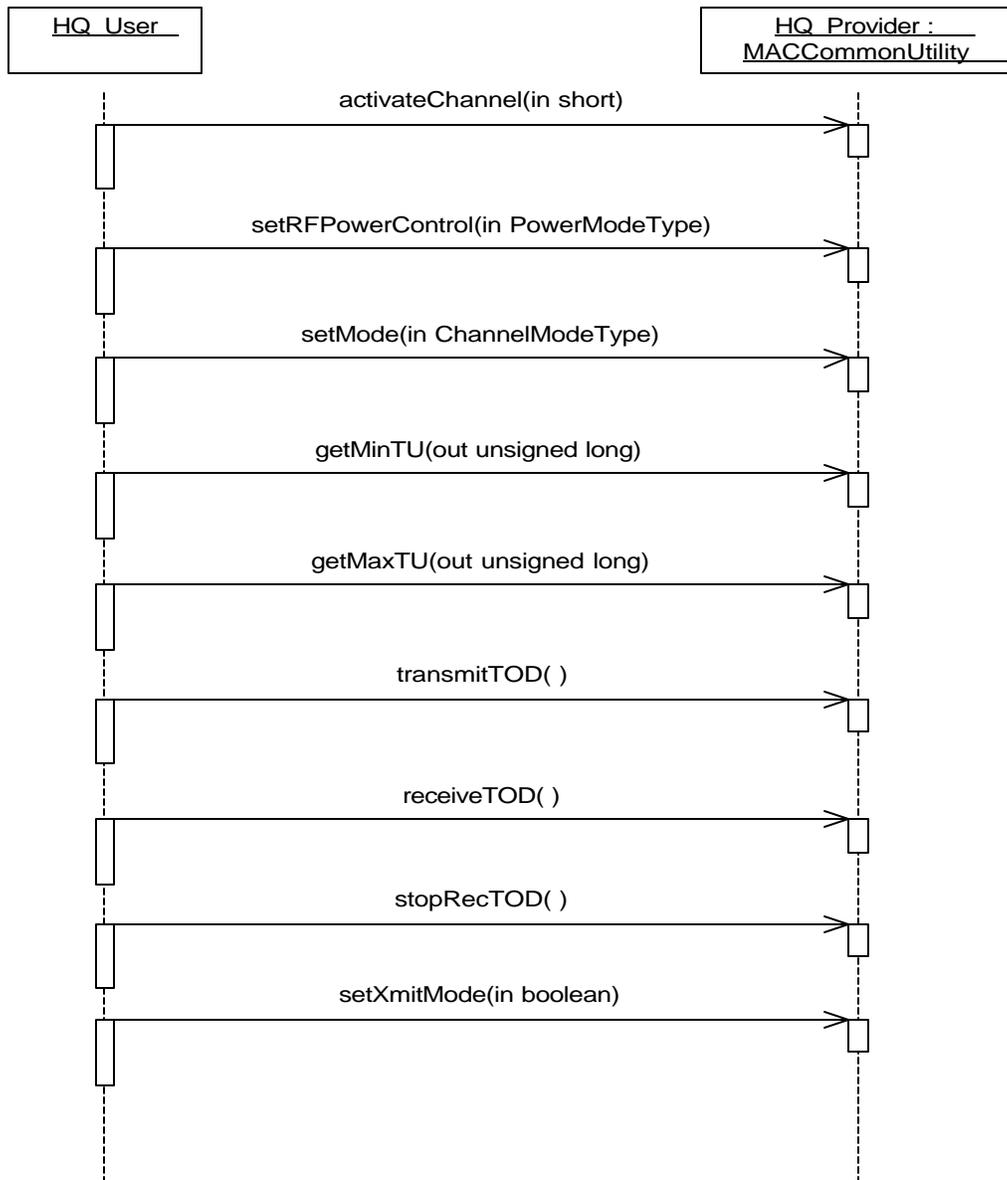


Figure 2. Sequence Diagram, MAC Common Utility

3.1.6 transmitTOD Service.

The transmitTOD Service initiates TOD packet transmission from this node.

3.1.7 receiveTOD Service.

The receiveTOD Service initiates a TOD reception mode at this node.

3.1.8 stopReceiveTOD Service.

The stopReceiveTOD Service terminates the ongoing TOD reception mode at this node.

3.1.9 setXmitMode Service.

The setXmitMode Service controls Push-To-Talk transmit activities.

3.2 TRANSEC.

The Transmission Security (TRANSEC) Service Group (Figure 3) provides a generalized interface to the MAC layer.

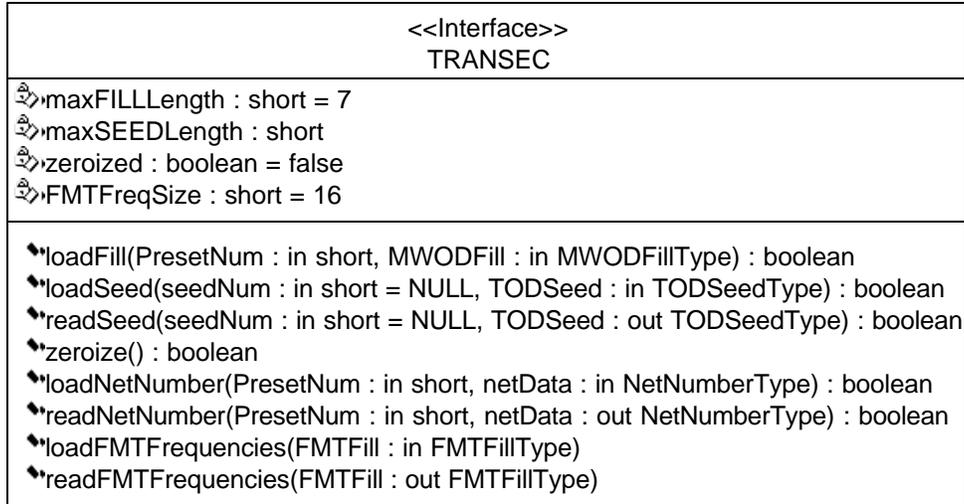


Figure 3. TRANSEC

3.2.1 loadFill Service.

The loadFill Service (Figure 4) provides a method to transfer MWOD data from the Service User to the MAC Layer. It returns the status of each transfer, Good or Bad.

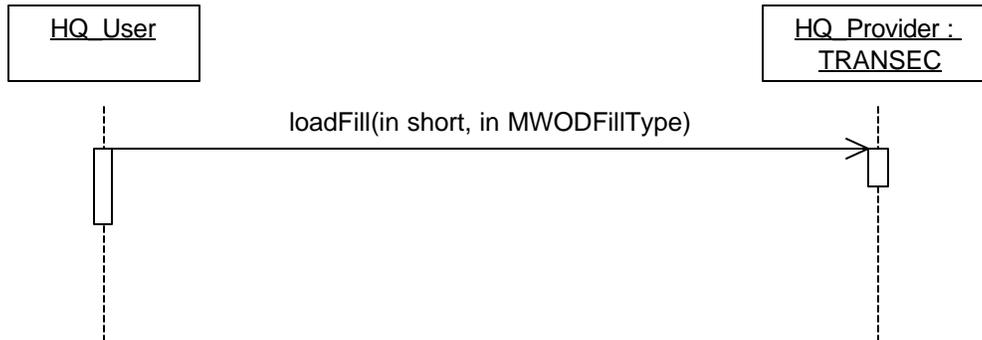


Figure 4. Sequence Diagram, loadFill

3.2.2 loadSeed Service.

The loadSeed Service (Figure 5) provides for transfer of TRANSEC TOD Seed data from the Service User to the MAC Layer.

3.2.3 readSeed Service.

The readSeed Service provides Service Users with a method of reading the actual TOD data.

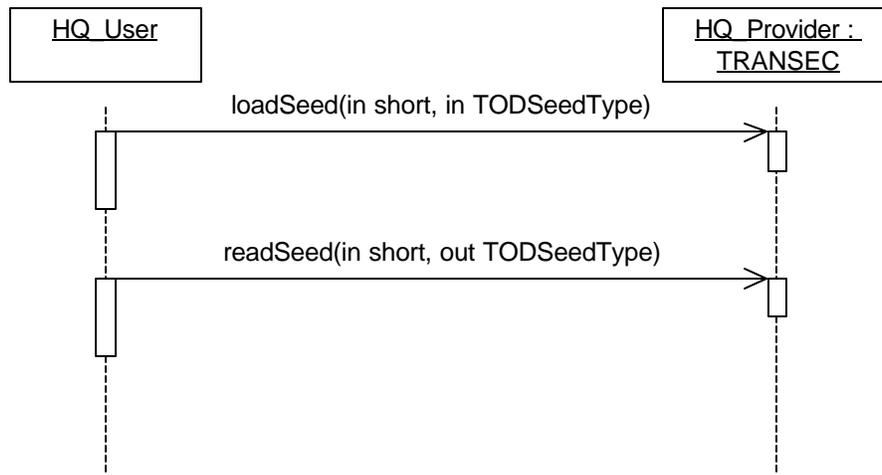


Figure 5. Sequence Diagram, loadSeed and readSeed

3.2.4 zeroize Service.

The zeroize Service (Figure 6) provides Service Users with a method of removing all Fill and Seed data from memory space owned by the TRANSEC BB. This service over-writes all Fill and Seed data memory space with a variety of data patterns to ensure information is not retained. This service does not release this memory space back to the Operating Environment.

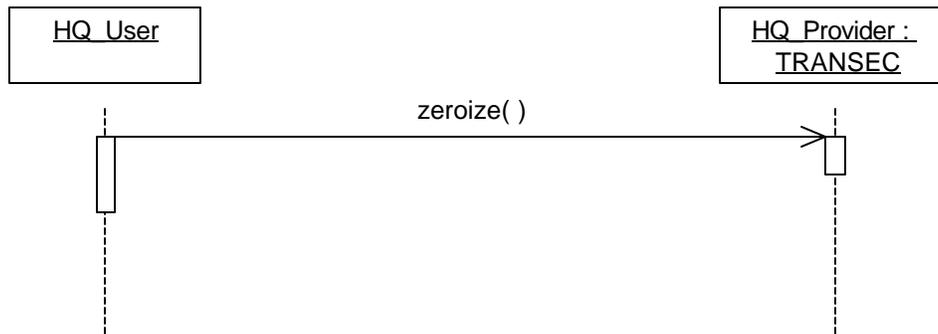


Figure 6. Sequence Diagram, zeroize

3.2.5 loadNetNumber Service.

The loadNetNumber Service (Figure 7) provides for transfer of net number information from the Service User to the MAC Layer.

3.2.6 readNetNumber Service.

The readNetNumber Service provides Service Users with a method of reading net number data.

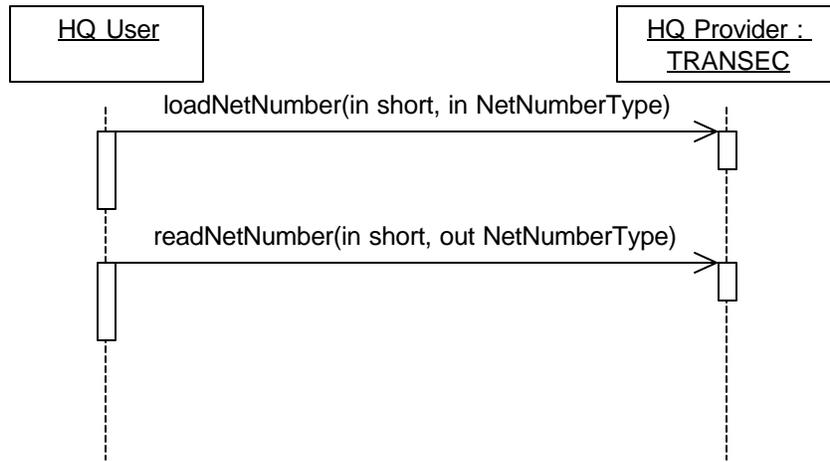


Figure 7. Sequence Diagram, loadNetNumber and readNetNumber

3.2.7 loadFMTFrequencies Service.

The loadFMTFrequencies Service (Figure 8) provides for transfer of FMT frequency data list information from the Service User to the MAC Layer.

3.2.8 readFMTFrequencies Service.

The readFMTFrequencies Service provides Service Users with a method of reading FMT frequency data.



Figure 8. Sequence Diagram, loadFMTFrequencies and readFMTFrequencies

3.3 CHANNELERRORCONTROL.

The ChannelErrorControl (Figure 9) attempts to maintain the integrity of the message over the physical channel.

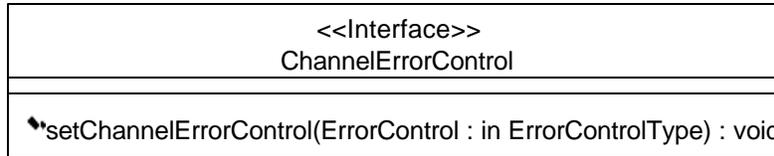


Figure 9. Channel Error Control

ErrorControlType is an enumerated type defined by the instantiating API Service Definition.

3.3.1 setChannelErrorControl Service.

The setChannelErrorControl Service (Figure 10) provides Service Users with a method to transfer non-real time control data.

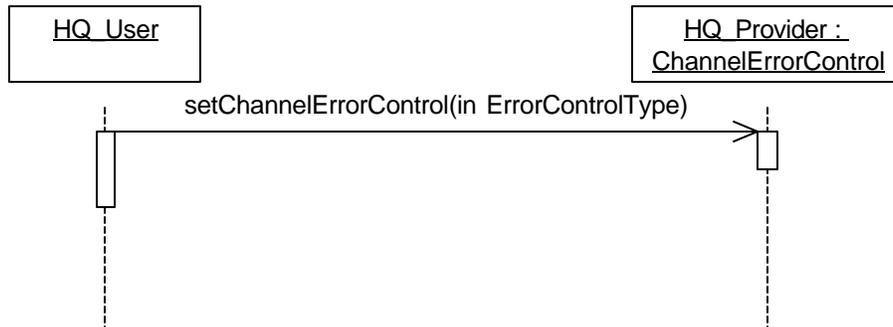


Figure 10. Sequence Diagram, setChannelErrorControl

3.4 ULONGPACKET.

The UlongPacket Service (Figure 11) provides the means to move data between layers. The Service User pushes a packet to the Service Provider and the Service Provider pushes a packet to the Service User.

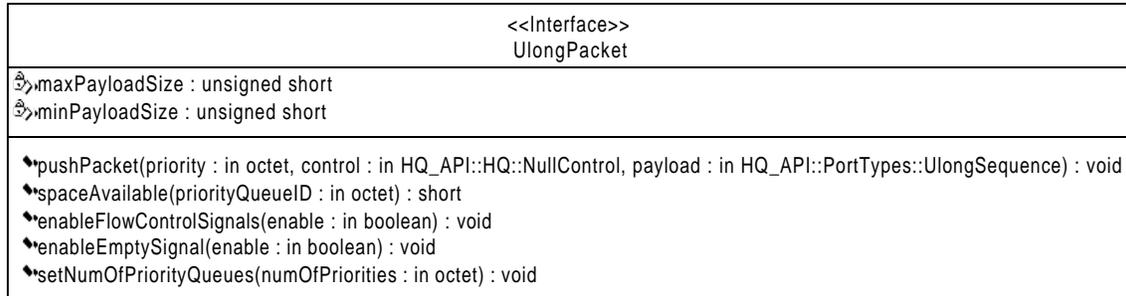


Figure 11. Data Packet

3.4.1 pushPacket Control Service.

The pushPacket Service (Figure 12) provides the Service User and the Service Provider with a method to transfer real-time data

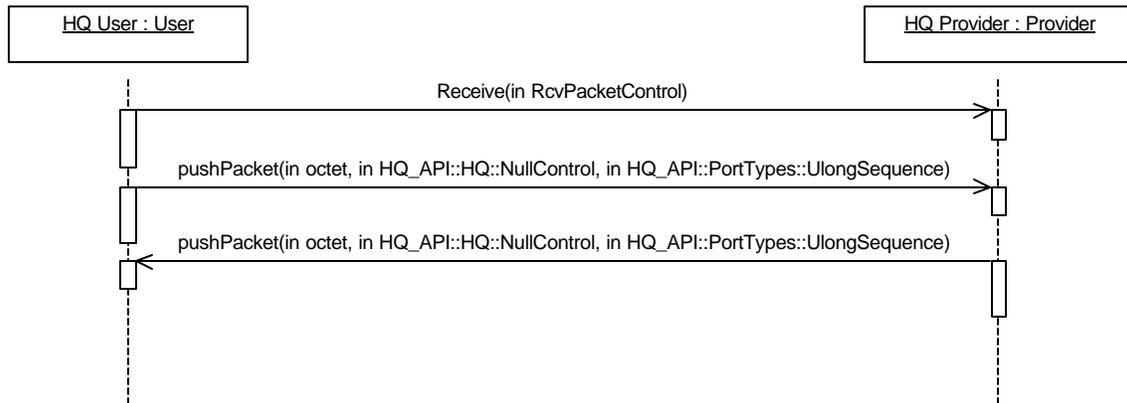


Figure 12. Sequence Diagram, Real-Time pushPacket

4 SERVICE PRIMITIVES.

4.1 MAC COMMON UTILITY SERVICE PRIMITIVES.

4.1.1 activateChannel Service.

"*activateChannel*" provides the ability to communicate to the active radio channel to the MAC Layer.

4.1.1.1 Synopsis.

boolean activateChannel (PresetNum : in short);

4.1.1.2 Parameters.

PresetNum : specifies the channel number or preset number of the active (selected) RF channel.
Short range 2^{16} .

4.1.1.3 State.

Any channel active.

4.1.1.4 New State.

New channel active.

4.1.1.5 Response.

This method returns True if the channel corresponding to the PresetNum is activated; otherwise, False is returned.

4.1.1.6 Originator.

Service User.

4.1.1.7 Errors/Exceptions.

Return = True if the channel is successfully activated; otherwise return = False.

4.1.2 setRFPowerControl Service.

"*setRFPowerControl*" provides the ability to communicate the desired RF power output level to the MAC Layer.

4.1.2.1 Synopsis.

boolean setRFPowerControl (PowerLevel : in PowerModeType)

4.1.2.2 Parameters.

PowerModeType: Parameter specifies RF power output level.

4.1.2.3 State.

Any Power Mode.

4.1.2.4 New State.

New Power Mode.

4.1.2.5 Response.

Return = True if the power control is successfully set with the given PowerMode; otherwise if Boolean = False.

4.1.2.6 Originator.

This primitive is initiated by the Service User.

4.1.2.7 Errors/Exceptions.

TBD

4.1.3 setMode Service.

"setMode" provides the ability to communicate the radio mode (e.g., FM, FH-CT-9600N, DASA, Wideband, etc.) to the MAC Layer.

4.1.3.1 Synopsis.

boolean setMode (ChannelMode : in ChannelModeType)

4.1.3.2 Parameters.

ChannelMode:

```
struct ChannelModeType {
    ModeConfigType ModeConfig;
    DataRateType DataRate;
    COMSECType COMSEC;
    OperationModeType OperationMode;
    DataModeConfigType DataModeConfig;
    Short ModemChannelNumber;
    Boolean transmitInhibit
};

enum ModeConfigType {Normal, Active };

enum DataRateType {D150, D300, D600, D1200, D2400, D4800, D75, DOFF, D9600N,
    PKT, D4800N, D2400N, D1200N, TF, D16000, RS232-9600,
    RS232-14400, RS232-38800};

enum COMSECType {PT, CT};

enum OperationModeType {LNE_Off, LNE_On, None};

enum DataModeConfigType {Audio, Data };
```

4.1.3.3 State.

Any Mode.

4.1.3.4 New State.

New_Mode.

4.1.3.5 Response.

Boolean return = True if the channel mode is successfully set; otherwise return is False.

4.1.3.6 Originator.

Service User.

4.1.3.7 Errors/Exceptions.

TBD

4.1.4 getMinTU Service.

"*getMinTU*" provides Service Users with the minimum Transmission Unit (TU) length the Service Provider will accept as being a valid packet..

4.1.4.1 Synopsis.

```
void getMinTU (MinTU : out unsigned long)
```

4.1.4.2 Parameters.

MinTU: inheriting API must declare the minimum length of a valid packet.

4.1.4.3 State.

Any state.

4.1.4.4 New State.

State remains unchanged.

4.1.4.5 Response.

unsigned long: range is 2^{32} . Value is maximum number of octets in a valid packet.

4.1.4.6 Originator.

Service User.

4.1.4.7 Errors/Exceptions.

TBD

4.1.5 getMaxTU Service.

"*getMaxTU*" provides Service Users with the maximum Transmission Unit (TU) length the Service Provider will accept as being a valid packet.

4.1.5.1 Synopsis.

```
Void getMaxTU (MaxTU : out unsigned long);
```

4.1.5.2 Parameters.

MaxTU: inheriting API must declare the maximum length of a valid packet.

4.1.5.3 State.

Any state.

4.1.5.4 New State.

State remains unchanged.

4.1.5.5 Response.

unsigned long: range is 2^{32} . Value is maximum number of octets in a valid packet.

4.1.5.6 Originator.

Service User.

4.1.5.7 Errors/Exceptions.

TBD

4.1.6 transmitTOD Service.

"*transmitTOD*" initiates transfer of TOD packet from this node.

4.1.6.1 Synopsis.

void transmitTOD (void);

4.1.6.2 Parameters.

None.

4.1.6.3 State.

Any state.

4.1.6.4 New State.

State remains unchanged.

4.1.6.5 Response.

None.

4.1.6.6 Originator.

Service User.

4.1.6.7 Errors/Exceptions.

TBD

4.1.7 receiveTOD Service.

"*receiveTOD*" initiates start of reception of TOD packet from this node.

4.1.7.1 Synopsis.

void receiveTOD (void);

4.1.7.2 Parameters.

None.

4.1.7.3 State.

Any state.

4.1.7.4 New State.

State remains unchanged.

4.1.7.5 Response.

None.

4.1.7.6 Originator.

Service User.

4.1.7.7 Errors/Exceptions.

TBD

4.1.8 stopRecTOD Service.

"*stopRecTOD*" initiates termination of reception of TOD packet from this node.

4.1.8.1 Synopsis.

```
void stopRecTOD (void);
```

4.1.8.2 Parameters.

None.

4.1.8.3 State.

Any state.

4.1.8.4 New State.

State remains unchanged.

4.1.8.5 Response.

None.

4.1.8.6 Originator.

Service User.

4.1.8.7 Errors/Exceptions.

TBD

4.1.9 setXmitMode Service.

"*setXmitMode*" signals a change in state for Push-To-Talk mode.

4.1.9.1 Synopsis.

```
boolean setXmitMode (PTTindicator : in boolean);
```

4.1.9.2 Parameters.

PTTIndicator.

This boolean signals PTT on or off.

4.1.9.3 State.

Any state.

4.1.9.4 New State.

State remains unchanged.

4.1.9.5 Response.

None.

4.1.9.6 Originator.

Service User.

4.1.9.7 Errors/Exceptions.

TBD

4.2 TRANSEC SERVICE PRIMITIVES.

4.2.1 loadFill Service.

"loadFill" provides a method to transfer TRANSEC MWOD data to the MAC Layer.

4.2.1.1 Synopsis.

```
boolean loadFill(PresetNum: in short, MWODFill : in MWODFillType);
```

4.2.1.2 Parameters.

PresetNum short range 2^{16} . Typically, associates fill data with a channel number.

MWODFill:

```
struct MWODFillType {  
    HQ_API::PortTypes::Ulong Sequence MWODData;  
};
```

4.2.1.3 State.

This service is available in any state except *zeroize* in progress.

4.2.1.4 New State.

MWODFillType loaded.

4.2.1.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.1.6 Originator.

Service User.

4.2.1.7 Errors/Exceptions.

TBD

4.2.2 loadSeed Service.

"loadSeed" provides the ability to transfer seeds such as Time-of-Day to the MAC Layer.

4.2.2.1 Synopsis.

boolean loadSeed(SeedNum: in short=Null,TODSeed: in TODSeedType);

4.2.2.2 Parameters.

SeedNum: Null

TODSeed:

```
struct TODSeedType {  
    HQ_API::HQ::DateType Date;  
    short Hours;  
    short Minutes;  
    short Seconds;  
    boolean Immediate;  
};
```

May be time-of-day, net time-of-day, word-of-day, etc.

4.2.2.3 State.

Any state except zeroize.

4.2.2.4 New State.

Seed (PresetNum) loaded.

4.2.2.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.2.6 Originator.

Service User.

4.2.2.7 Errors/Exceptions.

TBD

4.2.3 readSeed Service.

"readSeed" returns the actual Seed Time-of-Day (TOD) .

4.2.3.1 Synopsis.

string readSeed(SeedNum: in short, TODSeed: out TODSeedType);

4.2.3.2 Parameters.

SeedNum: Null

TODSeed:

```
struct TODSeedType {  
    HQ_API::HQ::DateType Date;  
    short Hours;  
    short Minutes;  
    short Seconds;  
    boolean Immediate;  
};
```

.

4.2.3.3 State.

Any.

4.2.3.4 New State.

No change.

4.2.3.5 Response.

Return *string* is the Seed data (e.g., TOD).

4.2.3.6 Originator.

Service User.

4.2.3.7 Errors/Exceptions.

TBD

4.2.4 zeroize Service.

"zeroize" provides a method of removing all Fill and Seed data from memory space owned by the TRANSEC BB.

4.2.4.1 Synopsis.

boolean zeroize ();

4.2.4.2 Parameters.

None.

4.2.4.3 State.

Any.

4.2.4.4 New State.

Fill (PresetNum = 0 to n) Empty and Seed (PresetNum = 0 to n) Empty

4.2.4.5 Response.

Boolean returns True when zeroization is successful; otherwise it returns False.

4.2.4.6 Originator.

Service User.

4.2.4.7 Errors/Exceptions.

TBD

4.2.5 loadNetNumber Service.

"*loadNetNumber*" provides the net number data for the specified preset to the MAC Layer.

4.2.5.1 Synopsis.

boolean loadNetNumber (PresetNum: in short, netData: in NetNumberType);

4.2.5.2 Parameters.

PresetNum: Indicates the specific preset number as an input.

NetData:

```
struct NetNumberType {  
    short netNumber;  
    short netType;  
};
```

May be time-of-day, net time-of-day, word-of-day, etc.

4.2.5.3 State.

Any.

4.2.5.4 New State.

No change.

4.2.5.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.5.6 Originator.

Service User.

4.2.5.7 Errors/Exceptions.

TBD

4.2.6 readNetNumber Service.

"*readNetNumber*" returns the net number data associated with the specified preset.

4.2.6.1 Synopsis.

string readNetNumber (PresetNum: in short, netData: out NetNumberType)

4.2.6.2 Parameters.

PresetNum: Indicates the specific preset number as an input.

NetData:

```
struct NetNumberType {  
    short netNumber;  
    short netType;  
};  
.
```

4.2.6.3 State.

Any.

4.2.6.4 New State.

No change.

4.2.6.5 Response.

Returns Net Number data as output.

4.2.6.6 Originator.

Service User.

4.2.6.7 Errors/Exceptions.

TBD

4.2.7 loadFMTFrequencies Service.

"loadFMTFrequencies" provides the FMT frequency list to the MAC Layer.

4.2.7.1 Synopsis.

boolean loadFMTFrequencies (FMTFill: in FMTFillType);

4.2.7.2 Parameters.

FMTFill:

```
struct FMTFillType {  
    HQ_API::PortTypes::UlongSequence FMTFrequencyData;  
};
```

4.2.7.3 State.

Any.

4.2.7.4 New State.

No change.

4.2.7.5 Response.

Boolean returns True if the operation was completed successfully; otherwise return is False.

4.2.7.6 Originator.

Service User.

4.2.7.7 Errors/Exceptions.

TBD

4.2.8 readFMTFrequencies Service.

"*readFMTFrequencies*" returns the FMT Frequency list data.

4.2.8.1 Synopsis.

string readFMTFrequencies (FMTFill: out FMTFillType)

4.2.8.2 Parameters.

FMTFill:

```
struct FMTFillType {  
    HQ_API::PortTypes::UlongSequence FMTFrequencyData;  
};
```

4.2.8.3 State.

Any.

4.2.8.4 New State.

No change.

4.2.8.5 Response.

Return *string* is the list of frequencies.

4.2.8.6 Originator.

Service User.

4.2.8.7 Errors/Exceptions.

TBD

4.3 CHANNELERRORCONTROL SERVICE PRIMITIVES.

4.3.1 setChannelErrorControl Service.

"*setChannelErrorControl*" provides for transfer non-real time control data to channel error control functions in the Mac Layer.

4.3.1.1 Synopsis.

void setChannelErrorControl (ErrorControl : in ErrorControlType);

4.3.1.2 Parameters.

ErrorControl :

```
enum ErrorControlType {
```

```
    Active,  
    Inactive  
};  
Used to enable/disable error control.
```

4.3.1.3 State.

Any.

4.3.1.4 New State.

Channel error control function(s) enabled, disabled or reconfigured.

4.3.1.5 Response.

None.

4.3.1.6 Originator.

Service User.

4.3.1.7 Errors/Exceptions.

TBD

4.4 ULONGPACKET SERVICE PRIMITIVES.

4.4.1 pushPacket Service.

"*pushPacket*" provides for transfer of real time date.

4.4.1.1 Synopsis.

```
void pushPacket(priority: in octet, control: in HQ_API::HQ::NullControl, payload: in  
HQ_API::PortTypes::UlongSequence)
```

4.4.1.2 Parameters.

Control :

```
struct NullControl {  
  
};  
typedef sequence <octet> OctetSequence;
```

Used to enable/disable error control.

4.4.1.3 State.

Any.

4.4.1.4 New State.

Same.

4.4.1.5 Response.

None.

4.4.1.6 Originator.

Service User.

4.4.1.7 Errors/Exceptions.

TBD

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

Any sequence of service primitives is allowable.

6 PRECEDENCE OF SERVICE PRIMITIVES.

There is no precedence of primitives.

7 SERVICE USER GUIDELINES.

There are no Service User guidelines for the HaveQuick MAC API.

8 SERVICE PROVIDER-SPECIFIC INFORMATION.

There is no Service Provider-Specific Information for the HaveQuick MAC API.

9 IDL.

The Have Quick interface design depicted in IDL source code is shown below.

```
//Source file: c:/program files/devstudio/vc/at1/include/HQ.idl

#ifndef __HQ_DEFINED
#define __HQ_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

module HQ {

    /* Control of packet for los with nothing in it. */

    struct NullControl {
    };

    struct DateType {
        long dayOfYear;
        long year;
    };

    module Physical {

        enum squelchType {
            AGC,
            NOISE
        };
    };
};
```

```
struct MediaParams_Type {
    squelchType squelchType;
    /* Gen_Set_Squelch_Msg #20
       Gen_AGC_Squelch_Msg #87 */
    unsigned long squelchLevelOffToOn;
};

interface MediaSetup {
    /*
       @roseuid 39F85AA6016C */
    boolean setUpMediaType (
        in MediaParams_Type MediaParams
    );
};

/* Provide a pushPacket with a octet payload */

interface OctetPacket {
    /* The maxPacketSize is a read only attribute set by the
       Packet Server and the get operation reports back the
       maximum number of traffic units allowed in one pushPacket
       call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the Server
       with a Control element and a Payload element.
       @roseuid 39F8739C037A */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in CF::OctetSequence payload
    );

    /* The operation returns the space available in the Servers
       queue(s) in terms of the implementers defined Traffic
       Units.
       @roseuid 39F8739C037E */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
       Watermark Signal ON and OFF.
       @roseuid 39F8739C0380 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty Signal
       ON and OFF.
       @roseuid 39F8739C0382 */
    void enableEmptySignal (
        in boolean enable
    );
};
```

```
/*
@roseuid 39F8739C0384 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);

};

/* Packet */

interface UlongPacket {
    /* The maxPacketSize is a read only attribute set by the
    Packet Server and the get operation reports back the
    maximum number of traffic units allowed in one pushPacket
    call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the Server
    with a Control element and a Payload element.
    @roseuid 39F875FC01E0 */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in PortTypes::UlongSequence payload
    );

    /* The operation returns the space available in the Servers
    queue(s) in terms of the implementers defined Traffic
    Units.
    @roseuid 39F875FC01FE */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
    Watermark Signal ON and OFF.
    @roseuid 39F875FC0208 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty Signal
    ON and OFF.
    @roseuid 39F875FC021C */
    void enableEmptySignal (
        in boolean enable
    );

    /*
    @roseuid 39F875FC0227 */
    void setNumOfPriorityQueues (
        in octet numOfPriorities
    );
};
```

```
};

/* Packet */

interface UshortPacket {
    /* The maxPacketSize is a read only attribute set by the
    Packet Server and the get operation reports back the
    maximum number of traffic units allowed in one pushPacket
    call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the Server
    with a Control element and a Payload element.
    @roseuid 39F87994028D */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in PortTypes::UshortSequence payload
    );

    /* The operation returns the space available in the Servers
    queue(s) in terms of the implementers defined Traffic
    Units.
    @roseuid 39F8799402AC */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
    Watermark Signal ON and OFF.
    @roseuid 39F8799402B5 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty Signal
    ON and OFF.
    @roseuid 39F8799402B7 */
    void enableEmptySignal (
        in boolean enable
    );

    /*
    @roseuid 39F8799402C0 */
    void setNumOfPriorityQueues (
        in octet numOfPriorities
    );
};

enum ModeType {
    Off,                /* HQ_Stop_Msg #6 */
    Operate,            /* HQ_Start_Msg #3 */
    Test                /* DSP_Run_Bit_Msg #34 */
};
```

```
/* Radio Mode */

interface RadioMode {
    /*
     * @roseuid 39F87E38030F */
    boolean setRadioMode (
        in ModeType RadioMode
    );
};

interface User : UlongPacket, PacketBB::PacketSignals {
};

/* This type is a CORBA unbounded sequence of octets. */
typedef sequence <octet> OctetSequence;

/* LOS modulation type */

enum ModulationType {
    AM
};

struct RcvPacketControl {
    /* The frequency to tune for the duration of the hop.. */
    unsigned long long frequencyInHz;
    /* Enumerated value that specifies the type of demodulation
     * or modulation to use . */
    ModulationType modulation;
};

/* Receive Command */

interface ReceiveCommand {
    /*
     * @roseuid 39F8815E01E1 */
    void Receive (
        in RcvPacketControl control
    );
};

interface Provider : ReceiveCommand, UlongPacket,
PacketBB::PacketSignals {
};

/* DSP_Capabilities_Msg #33 */

struct CapabilityType {
    boolean receive;
    boolean transmit;
    boolean sendModeMsg;
    boolean CVSD;
};
```

```

struct TransceiverParamsType {
    unsigned short transceiverID;
    /* Gen_Set_Frequency_Msg #23 */
    unsigned long frequency;
    CapabilityType capability;
};

/* Transciever Setup */

interface TransceiverSetup {
    /*
    @roseuid 39F87F0501A1 */
    boolean setUpReceiverParams (
        TransceiverParamsType RecvParams
    );

    /*
    @roseuid 39F87F0501BE */
    boolean setUpTransmitterParams (
        TransceiverParamsType TransParams
    );

    /*
    @roseuid 3A09AFE30395 */
    boolean setDate (
        in HQ::DateType date
    );

};

interface Controller : TransceiverSetup, MediaSetup, RadioMode,
CF::Resource {
};

};

module MAC {

    /* Mac octet Packet */

    interface OctetPacket {
        /* The maxPacketSize is a read only attribute set by the
        Packet Server and the get operation reports back the
        maximum number of traffic units allowed in one pushPacket
        call. */

        attribute unsigned short maxPayloadSize;
        attribute unsigned short minPayloadSize;

        /* This operation is used to push Client data to the Server
        with a Control element and a Payload element.
        @roseuid 39F998E602BA */
        void pushPacket (
            in octet priority,
            in HQ::NullControl control,
            in CF::OctetSequence payload
        );
    };
};

```

```
/* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic
Units.
@roseuid 39F998E602E3 */
short spaceAvailable (
    in octet priorityQueueID
);

/* This operation allows the client to turn the High
Watermark Signal ON and OFF.
@roseuid 39F998E602EC */
void enableFlowControlSignals (
    in boolean enable
);

/* This operation allows the client to turn theEmpty Signal
ON and OFF.
@roseuid 39F998E602EE */
void enableEmptySignal (
    in boolean enable
);

/*
@roseuid 39F998E602F7 */
void setNumOfPriorityQueues (
    in octet numOfPriorities
);
};

/* MAC unsigned long packet */
interface UlongPacket {
    /* The maxPacketSize is a read only attribute set by the
Packet Server and the get operation reports back the
maximum number of traffic units allowed in one pushPacket
call. */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the Server
with a Control element and a Payload element.
@roseuid 39F99B57021A */
    void pushPacket (
        in octet priority,
        in HQ::NullControl control,
        in PortTypes::UlongSequence payload
    );

    /* The operation returns the space available in the Servers
queue(s) in terms of the implementers defined Traffic
Units.
@roseuid 39F99B57022D */
    short spaceAvailable (
        in octet priorityQueueID
```

```
        );

    /* This operation allows the client to turn the High
    Watermark Signal ON and OFF.
    @roseuid 39F99B570237 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty Signal
    ON and OFF.
    @roseuid 39F99B570241 */
    void enableEmptySignal (
        in boolean enable
    );

    /*
    @roseuid 39F99B570243 */
    void setNumOfPriorityQueues (
        in octet numOfPriorities
    );
};

/* MAC unsigned short Packet */
interface UshortPacket {
    attribute unsigned short minPayloadSize;
    /* The maxPacketSize is a read only attribute set by the
    Packet Server and the get operation reports back the
    maximum number of traffic units allowed in one pushPacket
    call. */

    attribute unsigned short maxPayloadSize;

    /* This operation is used to push Client data to the Server
    with a Control element and a Payload element.
    @roseuid 39F99C500357 */
    void pushPacket (
        in octet priority,
        HQ::NullControl control,
        in PortTypes::UshortSequence payload
    );

    /* The operation returns the space available in the Servers
    queue(s) in terms of the implementers defined Traffic
    Units.
    @roseuid 39F99C500363 */
    short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High
    Watermark Signal ON and OFF.
    @roseuid 39F99C50036C */
    void enableFlowControlSignals (
        in boolean enable
```

```
        );

        /* This operation allows the client to turn theEmpty Signal
        ON and OFF.
        @roseuid 39F99C500375 */
        void enableEmptySignal (
            in boolean enable
        );

        /*
        @roseuid 39F99C500377 */
        void setNumOfPriorityQueues (
            in octet numOfPriorities
        );

};

interface User : UlongPacket, PacketBB::PacketSignals {
};

interface Provider : UlongPacket, PacketBB::PacketSignals {
};

enum ErrorControlType {
    Active,
    Inactive
};

/* MAC Channel Error Control */

interface ChannelErrorControl {
    /*
    @roseuid 39F9A91002BA */
    void setChannelErrorControl (
        in ErrorControlType ErrorControl
    );
};

struct PowerModeType {
    /* Gen_Set_Power_Msg #24 */
    short powerLevel;
};

enum DataRateType {
    D75,
    D150,
    D300,
    D600,
    D1200,
    D2400,
    D4800,
    D16000,
    D1200N,
    D2400N,
    D4800N,
    D9600N,
```

```

        DOFF,
        RS232-9600,
        RS232-14400,
        RS232-38800,
        PKT,
        TF
};

enum ModeConfigType {
    NORMAL,                /* Single Channel Mode
                           HQ_Normal_Msg #8 */
    ACTIVE                 /* Frequency Hop Mode
                           HQ_Active_Msg #7 */
};

enum OperationModeType {
    LNE_Off,
    LNE_On,
    None
};

enum COMSECType {
    PT,                    /* Gen_Plain_Text_Mode_Msg #32 */
    CT                     /* Gen_Cipher_Text_Mode_Msg #31 */
};

struct MWODFillType {
    PortTypes::UlongSequence MWODData;
};

struct TODSeedType {
    HQ::DateType date;
    short hours;
    short minutes;
    short seconds;
    boolean immediate;
};

struct NetNumberType {
    short netNumber;
    short netType;
};

struct FMTFillType {
    PortTypes::UlongSequence FMTFrequencyData;
};

/* MAC TRANSEC */

interface TRANSEC {
    attribute short maxFILLLength;
    /* Not used */

    attribute short maxSEEDLength;
    attribute boolean zeroized;
    attribute short FMTFreqSize;
};

```

```
/*
@roseuid 39F9C0E3026A */
boolean loadFill (
    in short PresetNum,
    in MWODFillType MWODFill
);

/*
@roseuid 39F9C0E3027E */
boolean loadSeed (
    in short seedNum,
    in TODSeedType TODSeed
);

/*
@roseuid 39F9C0E3030B */
boolean readSeed (
    in short seedNum,
    out TODSeedType TODSeed
);

/*
@roseuid 39F9C0E30315 */
boolean zeroize ();

/*
@roseuid 3A0846C20200 */
boolean loadNetNumber (
    in short PresetNum,
    in NetNumberType netData
);

/*
@roseuid 39F9C0E30274 */
boolean readNetNumber (
    in short PresetNum,
    out NetNumberType netData
);

/*
@roseuid 3A0966D501F8 */
boolean loadFMTFrequencies (
    in FMTFillType FMTFill
);

/*
@roseuid 3A0967020333 */
boolean readFMTFrequencies (
    out FMTFillType FMTFill
);

};

enum DataModeConfigType {
    Audio,          /* Gen_Audio_Mode_Msg #27 */
    Data            /* Gen_Data_Mode_Msg #28 */
};
```

```
struct ChannelModeType {
    ModeConfigType ModeConfig;
    DataRateType DataRate;
    COMSECType COMSEC;
    OperationModeType OperationMode;
    DataModeConfigType DataModeConfig;
    short ModemChannelNumber;
    boolean transmitInhibit;
};

/* Mac common utility */

interface MACCommonUtility {
    attribute unsigned long minTU;
    attribute unsigned long maxTU;

    /*
    @roseuid 39F9AA430303 */
    boolean activateChannel (
        in short PresetNum
    );

    /*
    @roseuid 39F9AA430316 */
    boolean setRFPowerControl (
        in PowerModeType PowerLevel
    );

    /*
    @roseuid 39F9AA430321 */
    boolean setMode (
        in ChannelModeType ChannelMode
    );

    /*
    @roseuid 39F9AA43032A */
    void getMinTU (
        out unsigned long MinTU
    );

    /*
    @roseuid 39F9AA43032C */
    void getMaxTU (
        out unsigned long MaxTU
    );

    /*
    @roseuid 3A0975A70009 */
    boolean transmitTOD ();

    /*
    @roseuid 3A0975A7006D */
    boolean receiveTOD ();

    /*
    @roseuid 3A0975A700C7 */
```

```
boolean stopRectOD ();

/*
@roseuid 3A09ACA80237 */
boolean setXmitMode (
    in boolean PTTindicator //Gen_PTT_Msg #76
);

};

interface Controller : CF::Resource, ChannelErrorControl,
MACCommonUtility, TRANSEC {
};

};

#endif
```

10 UML.

UML class and component diagrams are shown in Figures 13 through 20.

10.1 CONTROLLER DIAGRAM.

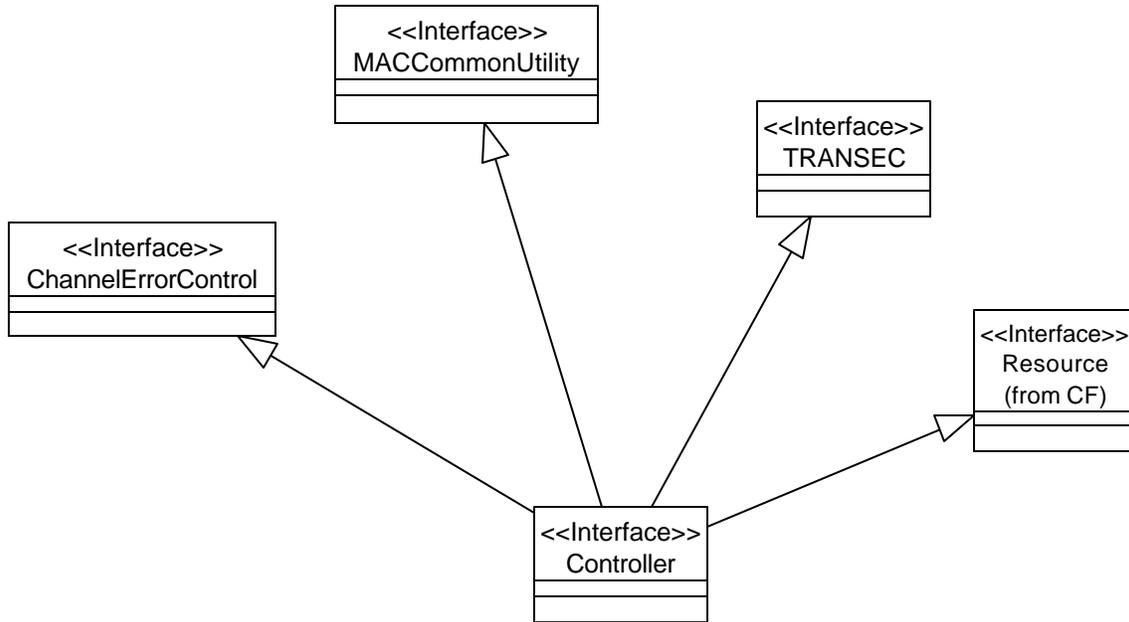


Figure 13. UML Controller Relationships

10.2 MAC COMMON UTILITY DIAGRAM.

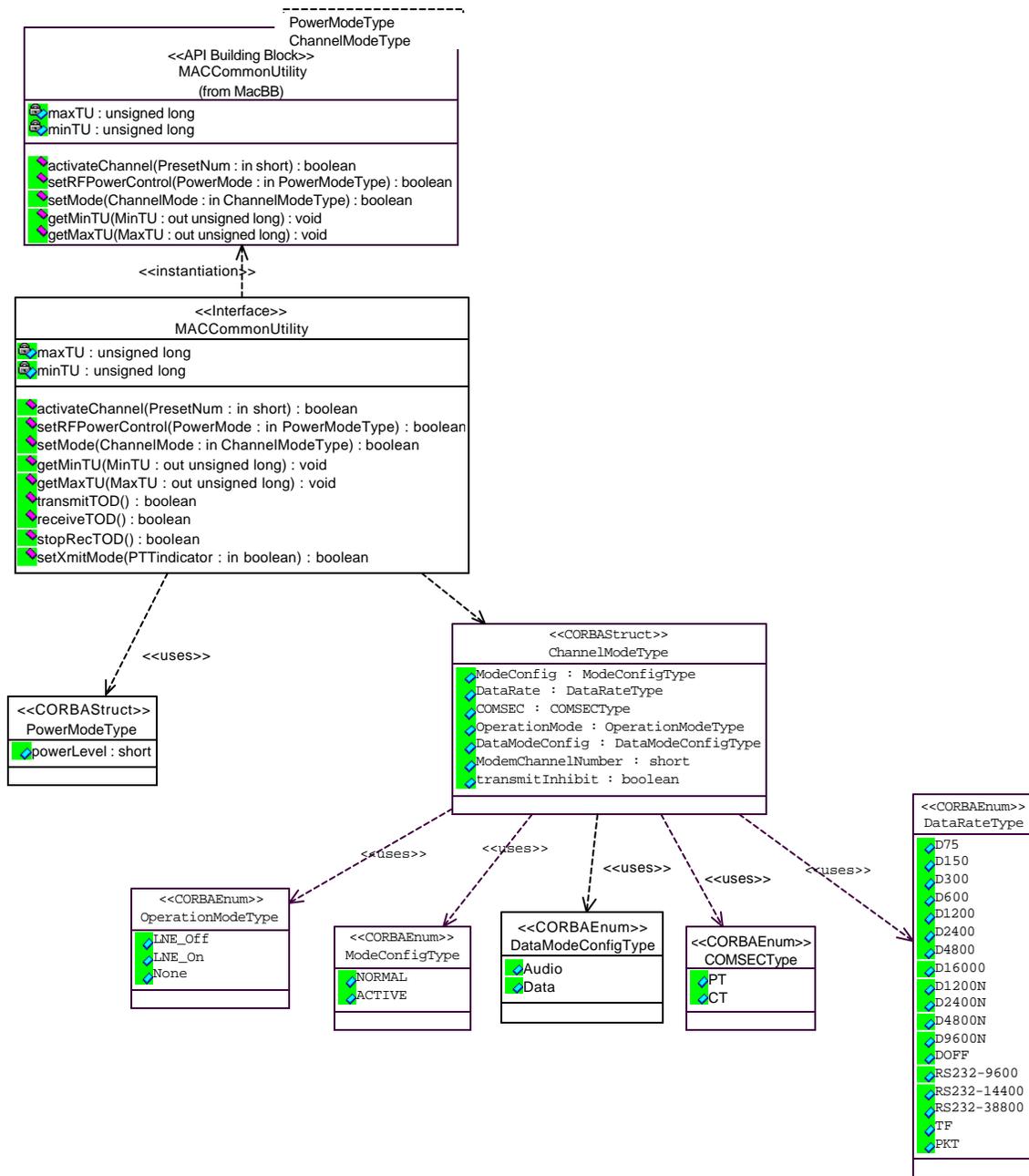


Figure 14. UML MAC Common Utility Relationships

10.3 TRANSEC DIAGRAM.

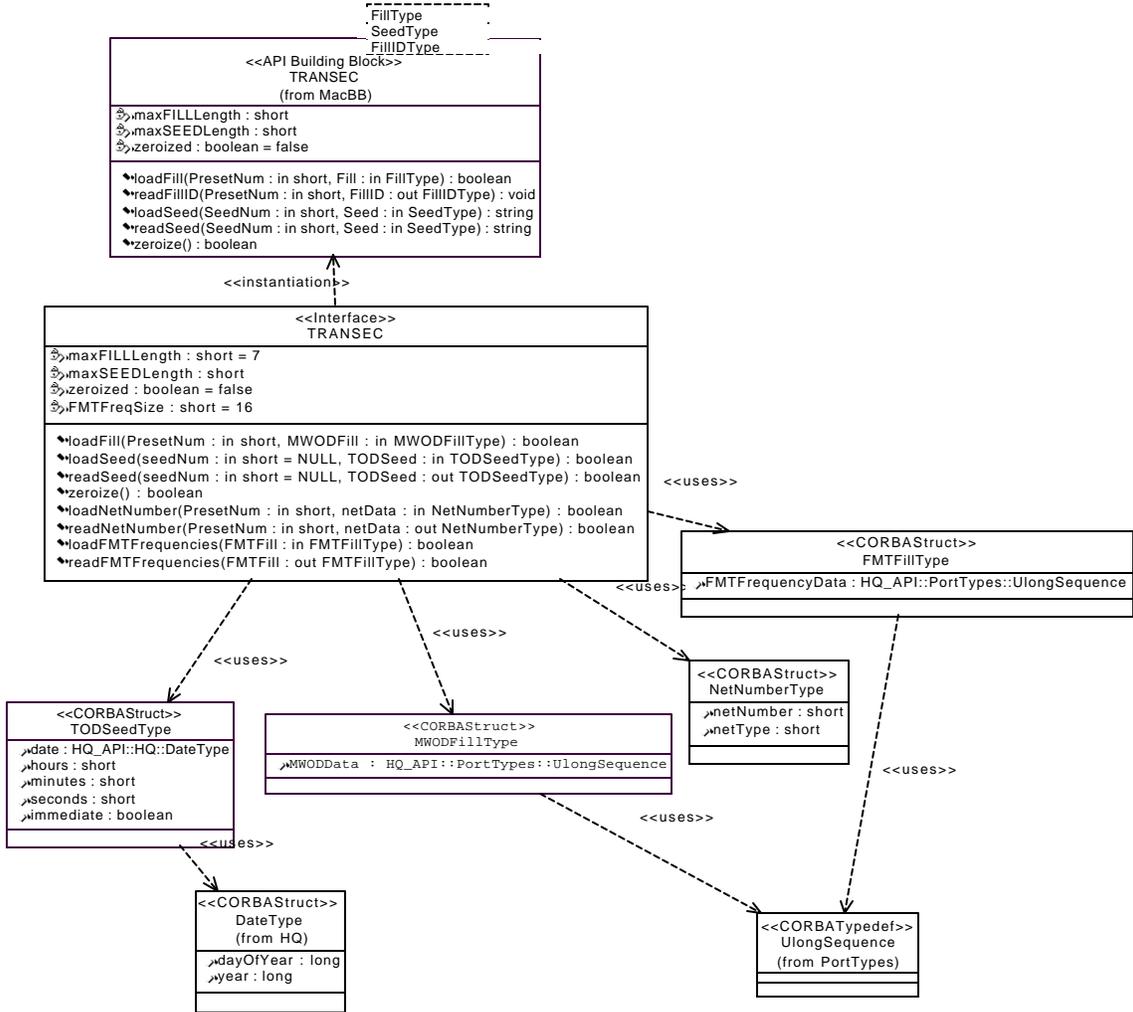


Figure 15. UML TRANSEC Relationships

10.4 CHANNELERRORCONTROL DIAGRAM.

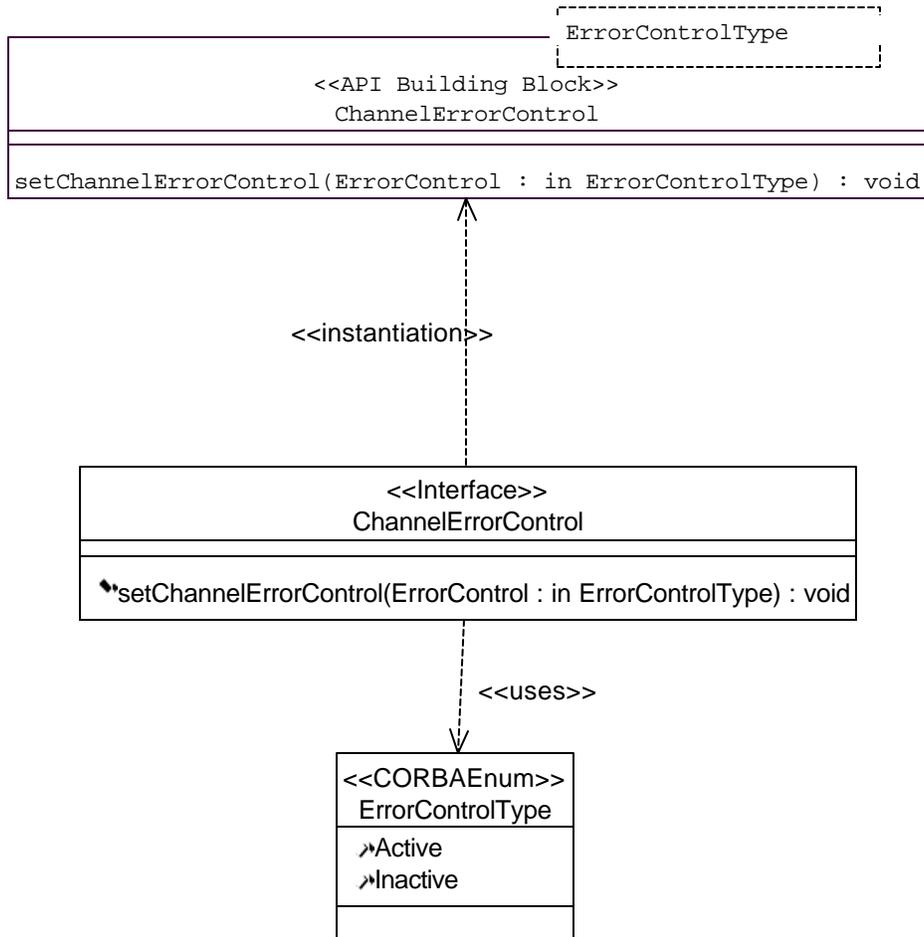


Figure 16. UML Channel Error Control Relationships

10.5 USER DIAGRAM.

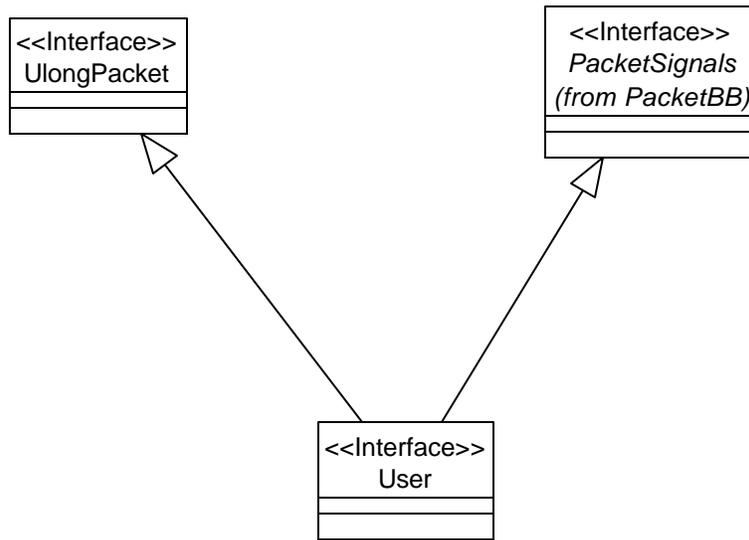


Figure 17. UML User, Data Packet Relationships

10.6 PROVIDER DIAGRAM.

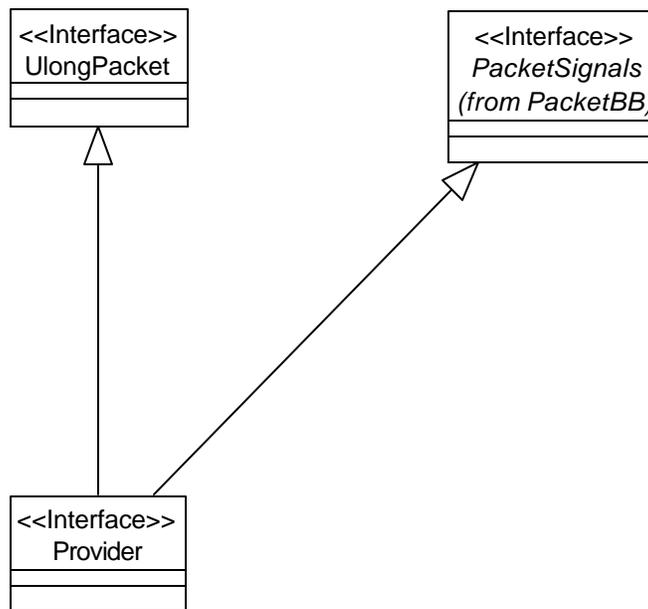


Figure 18. UML Provider, Data Packet Relationships

10.7 ULONGPACKET DIAGRAM.

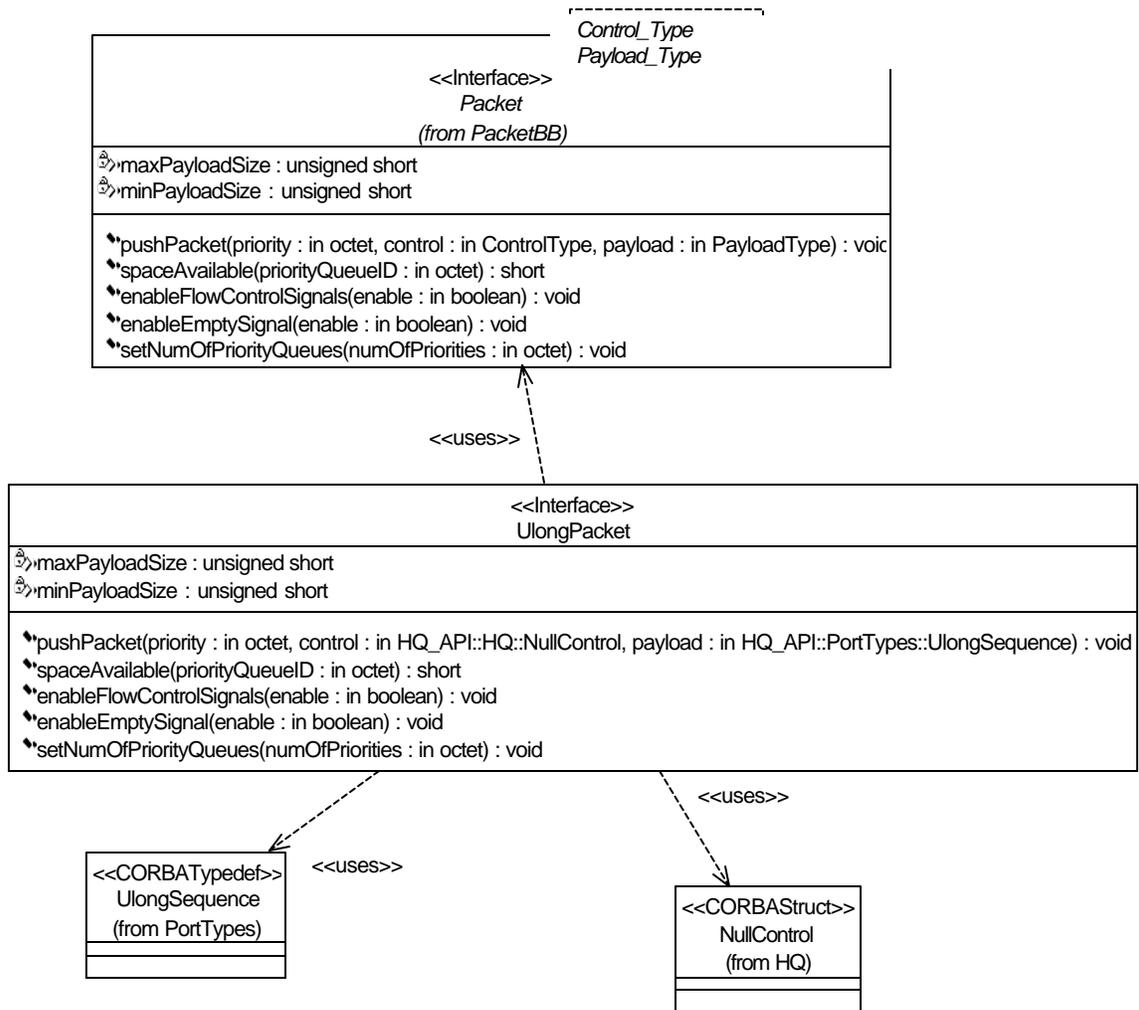


Figure 19. UML UlongPacket Relationships

10.8 COMPONENT DIAGRAM.

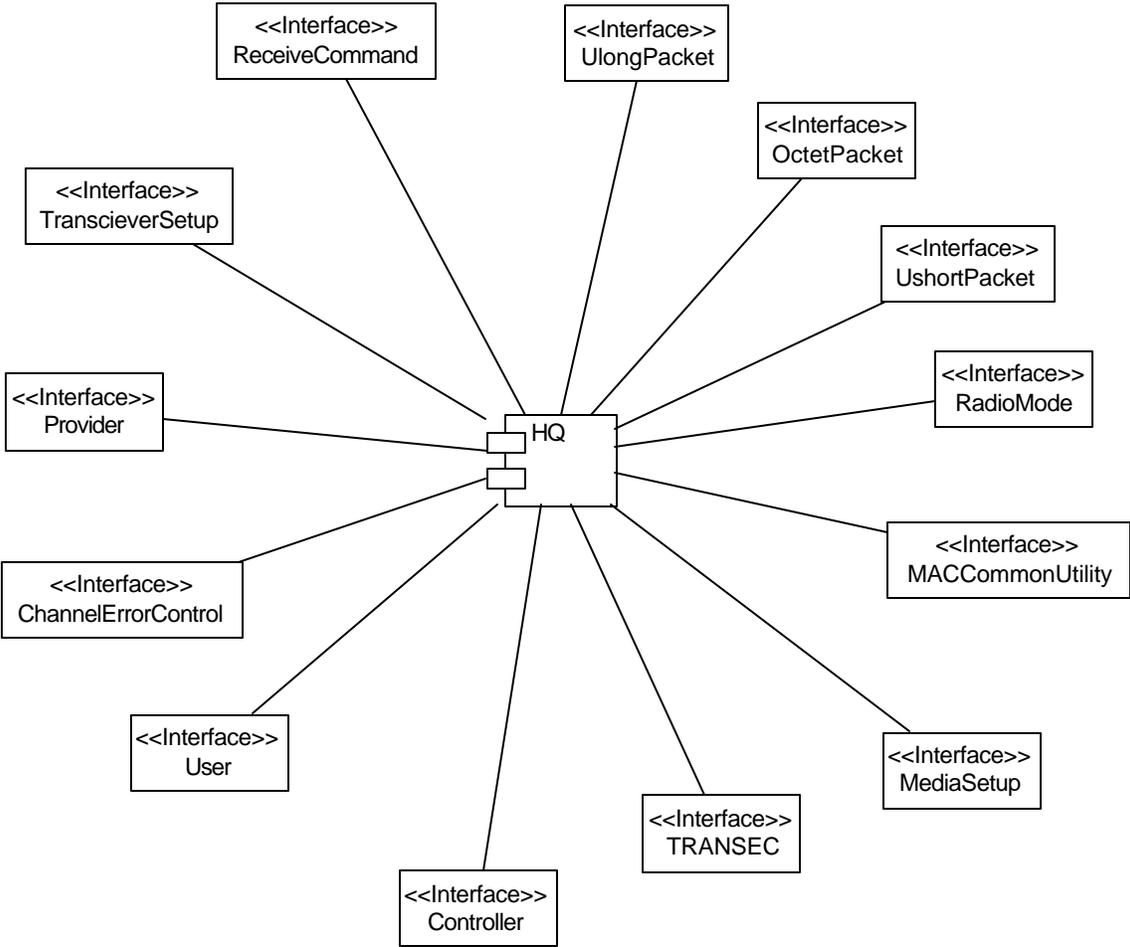


Figure 20. HQ Component Diagram